NP Completeness

Decision Problems

To keep things simple, we will mainly concern ourselves with decision problems. These problems only require a single bit output ``yes'' and ``no''.

How would you solve the following decision problems?

Is this directed graph acyclic?

Is there a spanning tree of this undirected graph with total weight less than w?

Does the pattern p appear as a substring in text t?

P

P is the set of decision problems that can be solved in worst-case polynomial time:

If the input is of size n, the running time must be O(n^k).
Note that k can depend on the problem class, but not the particular instance.

All the decision problems mentioned above are in P.

Nice Puzzle

The class NP (meaning non-deterministic polynomial time) is the set of problems that might appear in a puzzle magazine: ``Nice puzzle.''

What makes these problems special is that they might be hard to solve, but a short answer can always be printed in the back, and it is easy to see that the answer is correct once you see it.

Example... Does matrix A have an LU decomposition?

No guarantee if answer is ``no''.

NP

Technically speaking:

A problem is in NP if it has a short accepting certificate:
An accepting certificate is something that we can use to quickly show that the answer is ``yes'' (if it is yes).
Quickly means in polynomial time.
Short means polynomial size.

This means that all problems in P are in NP (since we don't even need a certificate to quickly show the answer is ``yes'').

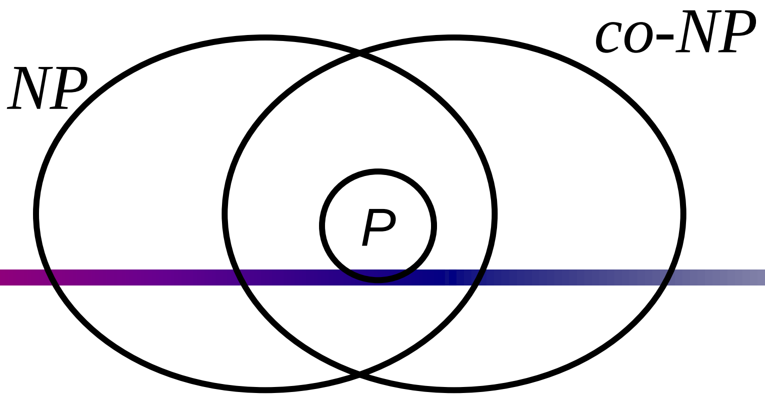
But other problems in NP may not be in P.

Is NP larger than P?

- Clearly, if a problem is in P it is also in NP. But what about the other way round?
- One might expect that such non-deterministic machines are more powerful (that is, that NP is larger than P).
- However, no one has found *a single problem* that is proven to be in NP but not in P.
- That is, if a problem is in NP, it might or might not be in P, so far as we know at present.
- In theory there *could be* efficient solutions to "hard" problems such as boolean satisfiability.

P=NP or P≠NP?

- Proving whether P=NP or P≠NP is one of the most important open problems in computer science.
- If someone showed that P=NP, then many "hard" problems (i.e. The NP-complete problems) would be tractable.
- However most computer scientists believe that P≠NP, largely because there are many problems which are in NP but for which no one has found an efficient solution.
  - That is, absence of evidence that P=NP counts as evidence that P≠NP.



One of the central (and widely and intensively studied 30 years) problems of (theoretical) computer science is to prove that

(a) ***P*** ✹ ***NP*** *(b)* ***NP*** ✹ ***co-NP***.

- *All evidence indicates that these conjectures are true.*
- *Disproving any of these two conjectures would not only be considered truly spectacular, but would also come as a tremendous surprise (with a variety of far-reaching counterintuitive consequences).*

NP-complete*: Collection Z of problems is NP-complete if (a) it is NP and (b) if polynomial-time algorithm existed for solving problems in Z, then P=NP.*

Some NP-complete problems

- Many practical problems are NP-complete.
  - Given a linear program (a set of linear inequalities) is there an integer solution to the variables?
  - Given a set of integers, can they be divided into two sets whose sum is equal?
  - Given two identical processors, a set of tasks of varying length, and a deadline, can the tasks be scheduled so that they finish before the deadline?
  - If there is an efficient solution to any of these, then all NP problems have efficient solutions! This would have a major impact.

NP-completeness:

- Class NPC:
- Some conclusions:
  1. if one NP-complete is solvable in polynomial time, then P = NP
  2. if P $\neq$ NP, then NPC $\neq \emptyset$
- Where or not P = NP is one of the most fundamental problems in CS.
- Since there are so many smart people who cannot solve the problem, if we are lazy then we show people the problems are NP-complete.

  No, showing NP-completeness doesn't end the story. We will see later.

Reduction

- The crux of NP-Completeness is *reducibility*
  - Informally, a problem P can be reduced to another problem Q if *any* instance of P can be "easily rephrased" as an instance of Q, the solution to which provides a solution to the instance of P
    - This rephrasing is called *transformation*
  - Intuitively: If P reduces to Q, P is "no harder to solve" than Q

Reducibility

- An example:
  - P: Given a set of Booleans, is at least one TRUE?
  - Q: Given a set of integers, is their sum positive?
  - Transformation: $(x_1, x_2, \ldots, x_n) = (y_1, y_2, \ldots, y_n)$ where $y_i = 1$ if x_i = TRUE, $y_i = 0$ if x_i = FALSE
- Another example:
  - Solving linear equations is reducible to solving quadratic equations

POLY-TIME *REDUCIBILITY*

A language A is polynomial time reducible to language B, written $A \leq_P B$, if there is a polynomial time computable function

$f : \Sigma^* \to \Sigma^*$, where for every w,

$$w \in A \Leftrightarrow f(w) \in B$$

f is called a polynomial time reduction of A to B

Using Reductions

- If P is *polynomial-time reducible* to Q, we denote this $P \leq_p Q$
- Definition of NP-Complete:
  - If P is NP-Complete, $P \in$ **NP** and all problems R are reducible to P
  - Formally: $R \leq_p P \ \forall\ R \in$ **NP**
- If $P \leq_p Q$ and P is NP-Complete, Q is also NP-Complete
  - This is the *key idea* you should take away today

NP-Hard and NP-Complete

- If P is *polynomial-time reducible* to Q, we denote this P $\le_p$ Q
- Definition of NP-Hard and NP-Complete:
  - If all problems R $\in$ **NP** are reducible to P, then P is *NP-Hard*
  - We say P is *NP-Complete* if P is NP-Hard and P $\in$ **NP**
  - **Note: I got this slightly wrong Friday**
- If P $\le_p$ Q and P is NP-Complete, Q is also NP- Complete

Why Prove NP-Completeness?

- Though nobody has proven that **P != NP**, if you prove a problem NP-Complete, most people accept that it is probably intractable
- Therefore it can be important to prove that a problem is NP-Complete
  - Don't need to come up with an efficient algorithm
  - Can instead work on *approximation algorithms*

- Question: How can we prove a problem to be NP-complete?
  Proof by definition?

  Yes, for the first NP-complete problem, we have to.
- After we know some NP-complete problems, we can prove a new problem L to be NP-complete through polynomial-time reduction:
  1. show $L \in$ NP
  2. find an NP-complete problem L'
  3. show $L' \leq_p L$

  We conclude that L is NP-complete.

(b) Let A and B be decision problems such that $A \leq_p B$. Answer true or false.

i. If $B \in \mathcal{P}$ then $A \in \mathcal{P}$. TRUE

ii. If B is $\mathcal{NP}$-complete then A is $\mathcal{NP}$-complete. FALSE

iii. If B can be decided in $O(n^3)$ time then A can be decided in $O(n^3)$ time.

FALSE

Given an undirected graph $G = (V, E)$, a set of vertices $W \subseteq V$ is a *clique* if for all $u, v \in W, (u, v) \in E$. *In other words, there is an edge between every pair of vertices in* W.

Given an undirected graph $G = (V, E)$, a set of vertices $W \subseteq V$ is an *independent set* if for all $u, v \in W, (u, v) \notin E$. *In other words, there is no edge between any pair of vertices in* W.

Consider the CLIQUE and INDEPENDENT SET (IS) problems for undirected graphs that we studied in class.

CLIQUE $= \{(G, k) \mid G$ has a clique of size $k\}$

IS $= \{(G, k) \mid G$ has an independent set of size $k\}$

We now define the new problem ISCLIQUE.

ISCLIQUE $= \{(G, k) \mid G$ has a clique of size k *and* an independent set of size $k\}$.

(a) Define a certificate for ISCLIQUE. Show that we can *verify* the certificate in deterministic polynomial time.

A certificate for ISCLIQUE is

2 sets $W_1, W_2 \subseteq V(G)$.

① for all $u, v \in W_1$, check if $(u, v) \in E(G)$. $O(n^2)$

② for all $u, v \in W_2$, check if $(u, v) \notin E(G)$. $O(n^2)$

③ check if $|W_1| = k$ $O(n)$

④ check if $|W_2| = k$ $O(n)$

(b) Consider an undirected graph G and an integer k. Construct a new graph H from G by adding k vertices to G but no additional edges. So, $G = (V, E)$ and $H = (V \cup W, E)$ where W is a set of k new vertices.

i. Show that if $(G, k) \in$ CLIQUE then $(H, k) \in$ ISCLIQUE.

Suppose $(G, k) \in$ CLIQUE. Let $C \subseteq V$ be the clique of size k in G. Since $C \subseteq V(H)$, C is also a clique in H. Also, W is an independent set of size k in H. So, H has a clique of size k and an independent set of size k, implying that $(H, k) \in$ ISCLIQUE.

ii. Show that if $(H, k) \in$ ISCLIQUE then $(G, k) \in$ CLIQUE.

Suppose $(H, k) \in$ ISCLIQUE. Let $C \subseteq V \cup W$ be a clique of size k in H. Since there are no edges incident on any vertex in W, no vertex in W is in C. So, $C \subseteq V$ and C is a clique of size k in G, implying that $(G, k) \in$ CLIQUE.

(c) What have we shown in part (a) and in part (b)? Using the fact that CLIQUE is NP-complete, what can we now conclude?

$$\left.\begin{array}{l} (a) \Rightarrow \text{ISCLIQUE} \in \mathcal{NP} \\ \\ (b) \Rightarrow \text{CLIQUE} \leqslant_p \text{ISCLIQUE} \end{array}\right\} \Rightarrow \begin{array}{c} \text{ISCLIQUE} \\ \text{is} \\ \mathcal{NP}\text{- complete} \end{array}$$